A Web Tool for Building Predictive Models for Alzheimer's Disease

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Annika Wille

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

December, 2023

A Web Tool for Building Predictive Models for Alzheimer's Disease

By Annika Wille

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Prof. Jason Sauppe Examination Committee Chairperson	Date
Prof. Samantha Foley Examination Committee Member	Date
Prof. Michael Petullo Examination Committee Member	Date

Abstract

Wille, Annika, "A Web Tool for Building Predictive Models for Alzheimer's Disease", Master of Software Engineering, November 2023, Dr. Jason Sauppe

This manuscript describes the design and development of a web tool for building predictive neural networks of Alzheimer's disease (AD). This web application allows users to learn more about the potential causes of the illness based on certain genetic markers, as well as allowing them to investigate their own SNP combinations in predictive models. Users are able to learn about various genetic markers that are associated with AD, and then build a model using specific markers of their choice. Approved research users can select up to five markers to compare and use as arguments for the predictive model. When the model has been built according to the user's specifications, various visualizations of results from the model are shared with researchers. This manuscript focuses specifically on design, development, testing, setbacks, and possible future work for this web application.

Acknowledgements

First, thank you to all of my professors within the Computer Science department for your support throughout the years. In particular, thank you to Dr. Jason Sauppe and Dr. Anne Galbraith, who provided essential insight regarding machine learning and genomics concepts. This project would not have been possible without them. I would also like to give a special thank you to Dr. Samantha Foley, who first introduced me to cross-disciplinary research and has been a huge inspiration throughout my time at UWL. Thank you to Dr. Petullo as well, whose mentorship regarding secure software development has been invaluable. Furthermore, I would like to thank Becky Yoshizumi, who always went above and beyond to help me pursue my passions and finish my degree. Finally, thank you to all of my friends and family, I do not know how I would have managed any of this without you. It took a village to get here, and I am so grateful for each and every single moment of support.

Table of Contents

List of Tables

List of Figures

Glossary

3' end

Refers to the directionality of a nucleic acid strand (DNA or RNA). The 3 prime end of a strand is that which has a free hydroxyl (or phosphate) on a 3' carbon.

5' end

Refers to the directionality of a nucleic acid strand (DNA or RNA). The 5 prime end of a strand is that which has a free hydroxyl (or phosphate) on a 5' carbon.

Amino Acid

The basic building block of proteins.

Codon

A group of three nucleotides which encodes a particular amino acid in a protein.

DNA

DNA is the molecule that carries genetic information in a cell, which determines an organism's traits and characteristics.

Exon

An exon is a segment of DNA within a gene that contains the coding information necessary to produce a functional protein or, in some cases, a functional RNA molecule. Exons are the parts of a gene that are transcribed into RNA during gene expression and are translated into a protein.

Gene

A segment of DNA within one's genome, the gene is considered the basic unit of inheritance. Genes are passed from parents to offspring and contain the information needed to specify physical and biological traits. Most genes code for specific proteins, or segments of proteins, which have differing functions within the body.

Gene Expression

Gene expression is the process by which information encoded in a gene's DNA is used to create functional products, such as proteins or RNA molecules, which carry out specific functions in a cell or organism.

Genome

The genome is the entire set of DNA (and genes) that exist in a cell.

Intron

An intron is a non-coding segment of a gene's DNA that does not contain instructions for producing a functional protein. Although they do not code for proteins, introns do play important roles in gene regulation, alternative splicing, and the evolution of genetic complexity.

Nucleotide

The basic building block of DNA (i.e. Adenine, Cytosine, Guanine, and Tyrine).

Neural Net Model

A neural network model is a computational model inspired by the human brain, consisting of interconnected nodes that process information to perform tasks like pattern recognition and data analysis. Neural net models learn from data and make predictions based on patterns and relationships within the data.

RNA

RNA, or ribonucleic acid, is a single-stranded copy (or transcription) of DNA.

Single Nucleotide Polymorphism (SNP)

An single nucleotide polymorphism, or SNP (pronounced 'snip') is a common type of genetic variation that occurs in DNA sequences. SNPs are variations of a single nucleotide in the genome that occur in a significant portion of the population.

Translation

Translation is the process in which proteins are produced using RNA molecules as templates

1. Introduction

1.1. Background

Alzheimer's disease (AD) is an illness that affects just under 6 million Americans presently, and this number is expected to only increase as time goes on [1]. Since AD can start developing in individuals almost 20 years before symptoms actually become visible, models that can help predict an individual's susceptibility to the disease early on are crucial. Early detection and prediction of AD allows for preventative action and medication before harmful components (e.g. tangles and plaques) have a chance to build up in the brain [9]. A significant amount of research has already been conducted studying the effect of individual genes on the prevalence of Alzheimer's disease. Unfortunately, there is no singular gene that has been found to directly cause AD. While certain genetic markers and mutations may increase risk, these individual events do not determine conclusively whether an individual will develop the disease.

Mutations on a single gene are only one of many ways in which diseases can be caused in the human body. Because genes interact in countless ways, many diseases can be caused by mutations in multiple genes (also known as multi-factorial disorders). Alzheimer's disease is an example of a multi-factorial disorder, however there is no combination of genetic mutations that is known to successfully predict AD thus far [2]. The goal of this project is to allow individuals to compare the effect of different combinations of genetic mutations as they relate to the prediction of Alzheimer's disease. More specifically, this project focuses on single nucleotide polymorphisms (SNPs) which are single point genetic mutations that exist within one's genome. A visual example of an SNP within a gene can be seen in Figure 1.

In order to fully understand the purpose of this web application, it is worth refreshing one's information on genes and proteins. A general overview of the route from genes to proteins can be seen in Figure 2, but will be discussed in depth below. Genes are basic units of DNA which make up the entire genome of a human being. Each gene is made up of countless nucleotides (Adenine, Guanine, Cytosine, and Thymine) and can be broken down into further subsections of introns and exons. Introns are non-coding regions of DNA, while exons are used to generate the RNA that is used to create proteins in the cells. Once RNA has been generated from the exons of a gene, that RNA is used as a template to create proteins from a series of amino acids. Amino acids are the base organic compounds which make up all proteins in a cell. Sets of three nucleotides known as codons correspond to specific amino acids which make up the protein. It is also worth noting that even if a DNA sequence isn't identical, it may result in the same amino acid as another sequence. While SNPs can occur within both introns and exons, mapping the consequences of mutations within an intron are difficult because they do not correspond with a specific protein structure or amino acid, but could impact the expression of other genes.

SNPs are known to have anywhere from negligible to severe effects within one's cells [10]. One SNP in a protein-coding gene might change a base pair in such a way that the resulting amino acid stays the same. Another might change an intermediate codon into a stop codon,

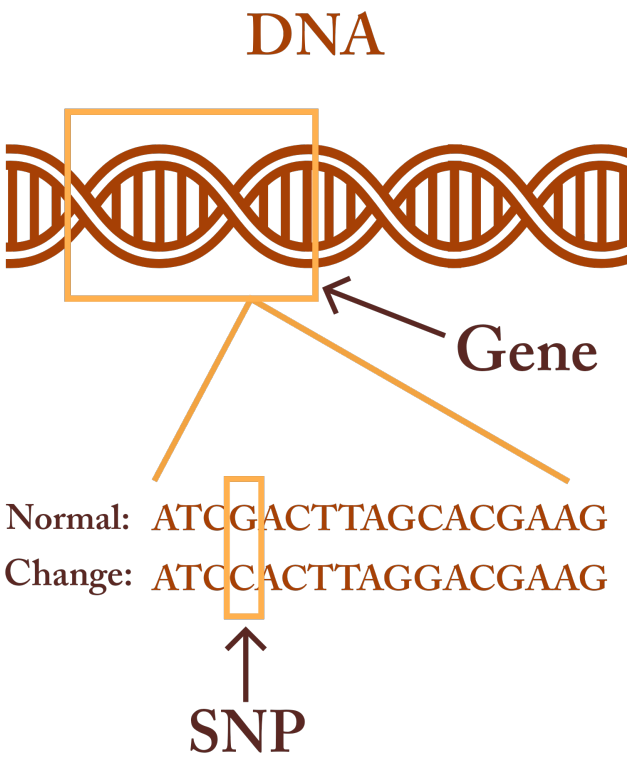

Figure 1. DNA, Gene and SNP diagram

which terminates protein coding abruptly and severely damages the resulting structure and function (Figure 3). In other cases, SNPs are used as genetic markers which can help identify genetic disorders, as well as hint at an individual's susceptibility to certain drugs or toxins [11]. Because SNPs can indicate an increased or decreased susceptibility to an illness (and because SNPs are quite common within genes), they can be used together to help predict the likelihood of someone becoming symptomatic of an illness such as AD.

Given the prevalence of SNPs, as well as their countless complex interactions, using them within machine learning models is an ideal way to spot patterns that otherwise might be overlooked. As mentioned earlier, this web tool focuses on allowing users to build predictive models for AD. This works by having users select up to 5 SNPs, and submitting them as a 'job'. On the back-end, the job will take those SNPs and use them as input arguments for a neural network model whose goal is to predict the likelihood of developing AD. Once the job is complete, results will be returned and the user will be able to see how well those SNPs worked at predicting AD. This allows users to investigate multiple SNP interactions to find potential markers and indicators of the illness.

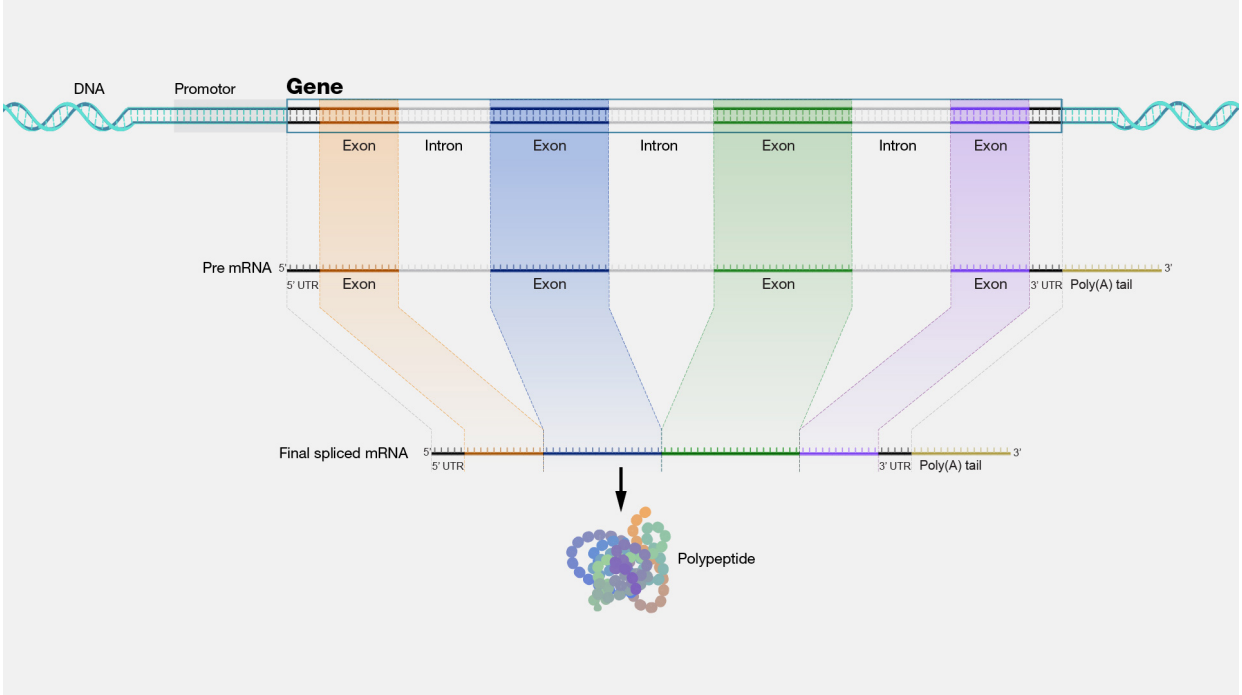

Figure 2. Gene to Protein Overview [8]

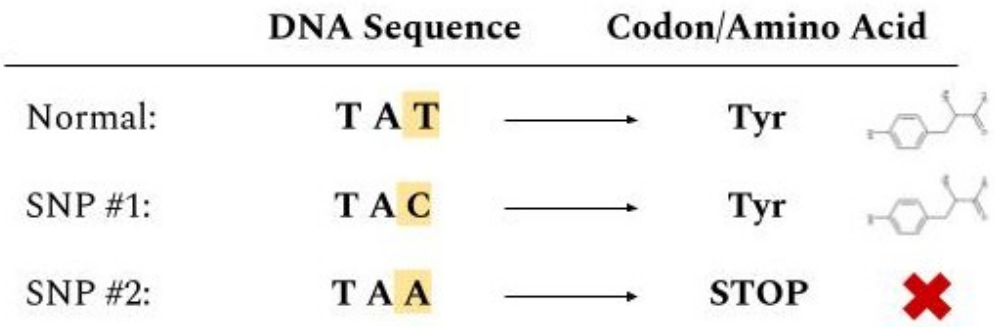

Figure 3. SNP Examples

1.2. Goals

This software product focuses on combining genetic ideas with machine learning in a simple web application. Given the complex nature of both the application and its purpose, one of the primary goals is to educate individuals on both the genetic and machine learning components of said application. Users should be able to research and inspect various genes and SNPs with ease so that they can draw their own conclusions about which indicators might be more successful in a predictive model. Furthermore, users should then be able to create, submit, and view the results of a job with ease. This web tool should also be designed such that results are easy to understand, and should hint at ways of improving upon the previously generated predictive model.

2. Requirements and Assumptions

2.1. Overview

Given the wide range of knowledge spanned in this web application, experts in both genomics and machine learning were consulted to ensure that any gaps in understanding were bridged sufficiently. In particular, five college students with biological and genomic areas of study were consulted throughout the process of generating the web tool requirements. These particular individuals were interviewed because they are representative of ideal users of such a tool. Taking into consideration both usability and security of the product, the following requirements were determined:

2.1.1. Non-functional Requirements

- The web application must not allow any users, approved or unapproved, to access genomic data used in both testing and training of the neural network
- The user interface should be intuitive and user-friendly.
- The web application should handle any errors gracefully.

2.1.2. Functional Requirements

- Users can select a variety of genes to compare their effect on the prevalence of AD
- Users can select a variety of SNPs to compare their effect on the prevalence of AD
- Information about each gene and its particular function and expression should be available to researchers who do not have a strong biology background
- Information about SNPs including their locations and potential consequences should be available to researchers who do not have a strong biology background
- Information about machine learning options for developing neural network models should be easy to follow for researchers who do not have a strong machine learning background
- Allow users to have default values for developing neural network models for users who don't care about the neural network used
- Allow experienced users the ability to customize neural network models if they have a grasp of various machine learning concepts and ideas of what might work most effectively
- Approved users must be verified with a .edu email in order to create and login to a privileged account
- Users should be emailed to securely login to see results when neural network training and evaluation is finished

- Neural network performance should be displayed in easy to understand visual diagrams for users with a primarily biological background, with final prediction scores
- Suggestions for improving neural network performance should be visible to users if their model performance is low
- Users may only submit one job at a time, but can set up a queue of jobs that they would like to have processed
- Neural network configurations and result information can be saved and downloaded as a pdf for the user who wishes to export the performance results
- Glossary definitions for basic genetic and machine learning terms will be available to users for further clarification of specific concepts and terms

2.2. Assumptions

As discussed earlier, targeted users for this web tool are primarily biology researchers who already have a background in genomics. Ideally, the researchers will have some experience studying Alzheimer's disease, however that is not necessarily a prerequisite to using this web application. Because one of the primary purposes of this web tool is to educate individuals on both genomic and machine learning concepts, there are very few assumptions about users of this product. However, it is worth noting that there are some basic biological concepts that users are assumed to have familiarity with, such as the general ideas with translation, transcription, and protein production within cells. The few assumptions that are relevant to this application are included below:

- All researchers will be connected to some university system and thus have access to an .edu email account
- All users will have a basic (high school level) knowledge of genetics

3. Design

3.1. Development Approach

One aspect of this project worth noting is the unknown annotations and attributes of data that will be implemented in this web application. Because it is uncertain exactly how many attributes will be consistently updated and annotated with various SNPs, it is important to utilize an approach throughout development that allows for maximum flexibility. Furthermore, the initial goal is to use real genomic data for the neural network models if possible. However, it is worth recognizing that the possibility for obtaining said data is slim, and thus artificial data may need to be generated instead. This inherent uncertainty built into the project makes it clear that Agile is the ideal development approach.

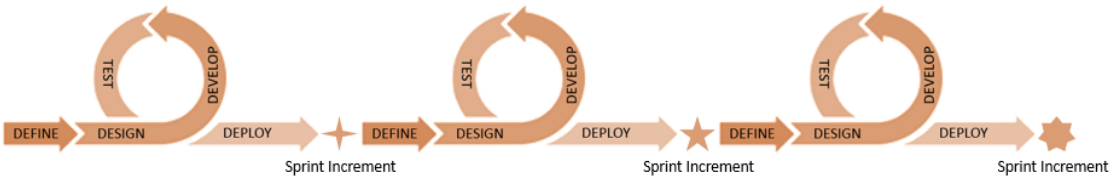

Figure 4. Agile Development Model [6]

Rather than a typical waterfall model, which consists of one larger, comprehensive iteration of five development phases, Agile development consists of numerous short iterative cycles known as "sprints". These shorter sprints allow for changing requirements to be implemented in an efficient manner, which is perfect for this web tool. Another benefit of using the Agile approach is the inherent focus on communication that is built into the methodology. Prior to each sprint, it is important to make sure that not only are biologists going to be able to use the tool, but computer scientists as well. To make sure that each respective group can understand both sides of the application, there will need to be an emphasis on communication with individuals in both fields throughout the entirety of development. The Agile methodology lends itself to this communication requirement, because every sprint begins with discussing which requirements should be implemented next.

Within the Agile methodology, each sprint cycle can be broken down into the following five stages: requirements gathering, design, implementation, testing, and deployment. Requirements gathering, the first step of each sprint cycle, involves identifying which requirements should be prioritized and implemented first. Usually, this stage involves lots of collaboration and feedback from targeted users of the project, to ensure that the desired functionality is achieved. Once the priority requirements have been identified and selected for the next sprint, the next step of the cycle is the planning stage. During the planning or design phase of a sprint cycle, smaller sprint tasks for each requirement are generated to further break up the workload into manageable segments. Examples of relevant sprint tasks to this project will be discussed later in the implementation section of this paper, but a quick generic example of a sprint task might be something such as: Design an endpoint for user authentication

on the back-end of the Django application. The goal of sprint tasks is that they should be able to be completed in a day or so. Sprint tasks are also useful when generating test cases later on in the sprint.

Once sprint requirements have been identified and tasks have been planned out, it is time to implement each sprint task. If any unforeseen tasks arise that cannot be completed within the current sprint cycle, they should be noted and prioritized for a later cycle. After successfully implementing sprint tasks, testing should occur next. In sprint cycles for this project, many of the test cases were generated in conjunction with specific sprint tasks. This helps simplify the testing process and identify any potential bugs quickly. Once the testing stage of the sprint cycle is completed, it is finally time to deploy all new updates to the project (if applicable) and review how the previous sprint went before starting the next cycle. Throughout this project, small changes were made to this sprint cycle structure, but overall the general steps described here were followed during each cycle.

3.2. Design Overview

In order for this web tool to work successfully, there were two major components that were designed prior to development. First, databases for both user authentication and genetic data storage were designed in advance given the massive amount of data necessary for this application. In addition, the user interface via the web application was designed initially to provide a sense of structure, although this initial design was improved upon as sprints went on. The overall architecture of the system can be seen in Figure 5.

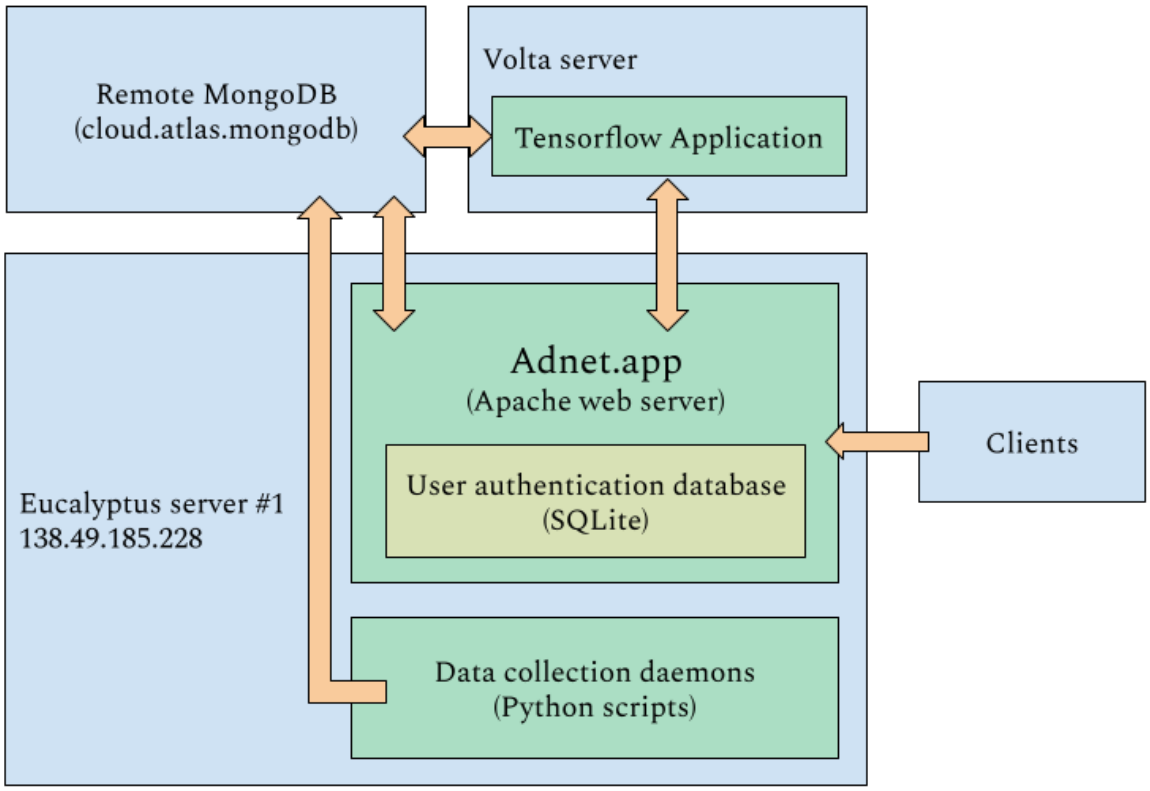

Figure 5. Architecture Diagram

As shown in Figure 5, Eucalyptus server 1 hosts the main Django web application via an Apache server. Within the Django application is a built-in SQLite database which works well for user authentication. Once collected, a remote Mongo database is updated with any new or changed information. The AdNet application uses data from both the remote and SQLite databases to display information to clients. Finally, clients can submit jobs through the Apache web server to be sent to the TensorFlow application (which is hosted on the remote volta server for better performance). Once the results have been processed and a neural network model has been produced, the TensorFlow application returns those results back to the AdNet application for clients to view. Further specification on the design of these components such as the databases are discussed below.

3.3. Database Design

As mentioned earlier, flexibility throughout the project is essential. Given both the uncertainty of genomic data that will be collected, as well as the unknown size of said data, a remote Mongo database seems ideal for this aspect of the project. Although initially the Mongo database couldn't be designed due to the unknown variables in the data, once data collection on genes and SNPs began, the following design in Figure 6 was created and implemented.

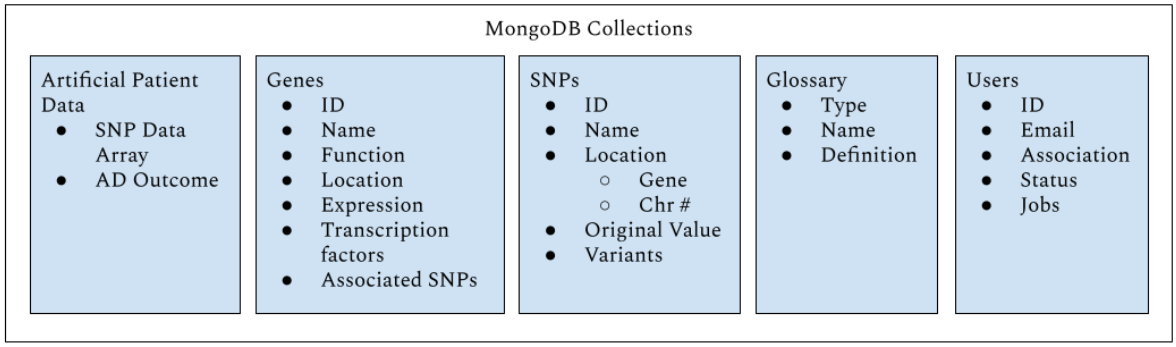

Figure 6. Initial MongoDB Design

During a later sprint more data sources were implemented to increase essential information, such as an SNP's strongest risk alleles, and associated studies. As a result, the Mongo documents were redesigned to incorporate this new information, as seen in Figure 7.

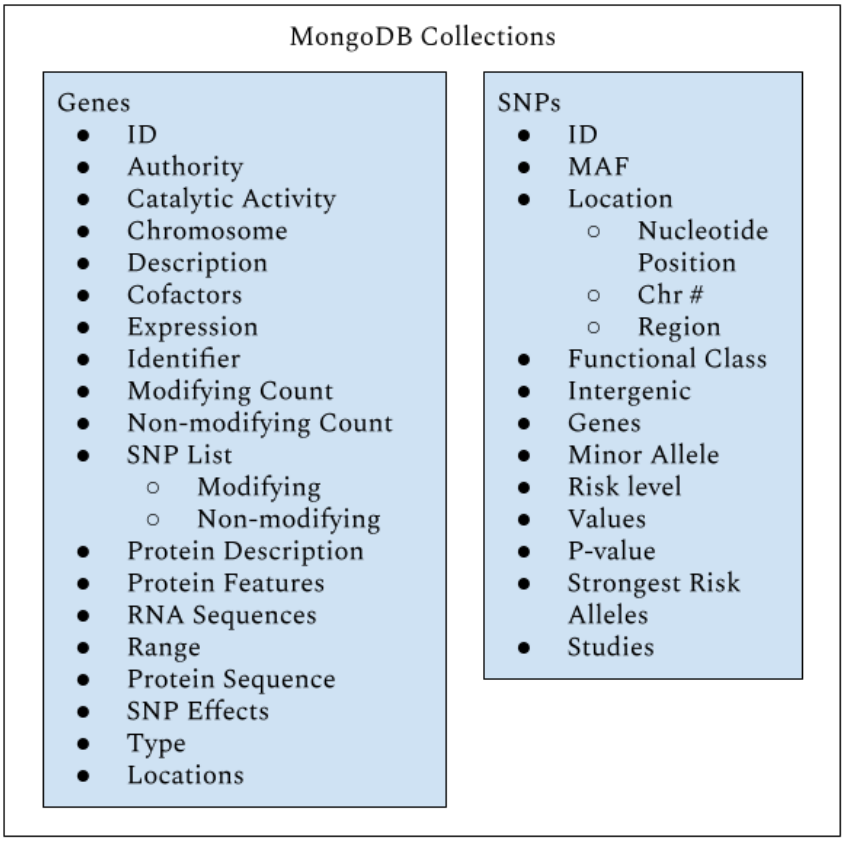

Figure 7. Revised MongoDB Design

MongoDB is not the only database which will be used in this web tool. A smaller SQLite database, which is built into the Django application, will be used for user authentication and

account creation. This will simplify the scope of the project and allow for increased security with the web server. Given the costly time and computational resources used to create a predictive neural network model, only authenticated users with a verified '.edu'email will be allowed to create an account and submit jobs. SQLite allows for user permissions to be set and assigned upon both account creation and after email verification. Fields in this simple database include a user's first name, last name, email, and institution, in addition to an encrypted password.

3.4. User Interface Design

The goal of the interface design is simplicity. As a result, a single template with multiple tabs was decided upon. Thankfully, Django has built in functionality to help reach this objective. For the web tool, there are two main "views" that a user can have of the home page, depending on their authentication status. The non-authenticated view displays genes, SNPs, and glossary terms, but does not allow for the submission of jobs. The authenticated view is where verified users can submit and process various jobs to create neural network models. An initial design of both views can be seen in Figures 8 and 9.

The design of each tab within the page will not be determined at the start of the project, due to the unknown data attributes. Instead, tab page designs (i.e. Gene Search) will be created at the start of an appropriate and related sprint cycle. An example of a page design that occurred at the start of a sprint can be seen in Figure 10.

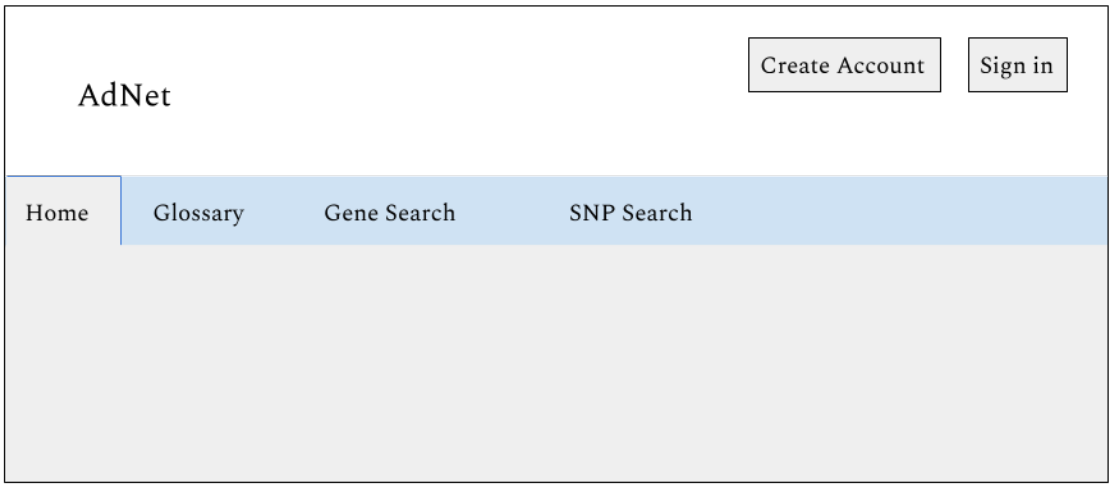

Figure 8. Non-authenticated View Design

3.5. Neural Net Design

As mentioned in the requirements section, one of the goals for users of this project is that they should be able to modify a neural net model to fit their specifications. That being said,

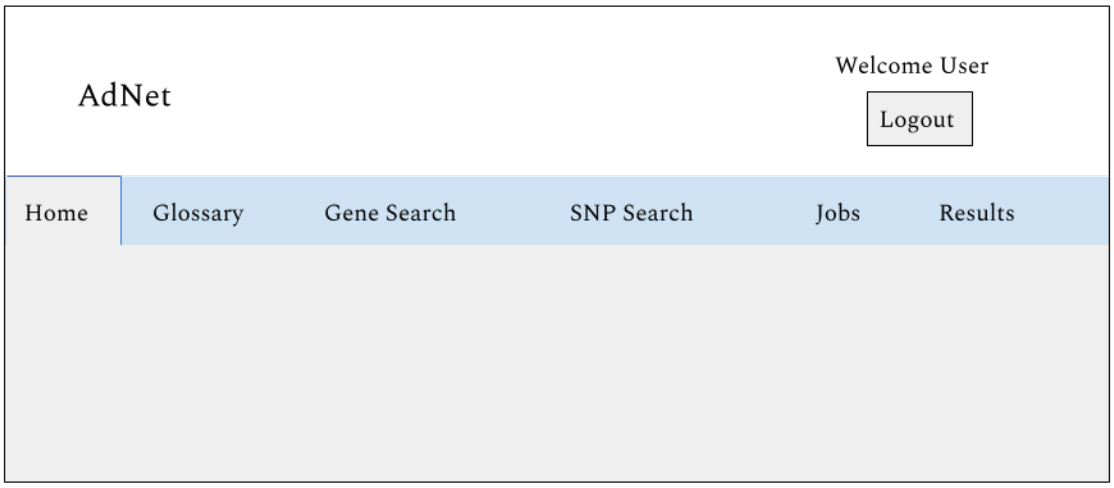

Figure 9. Authenticated View Design

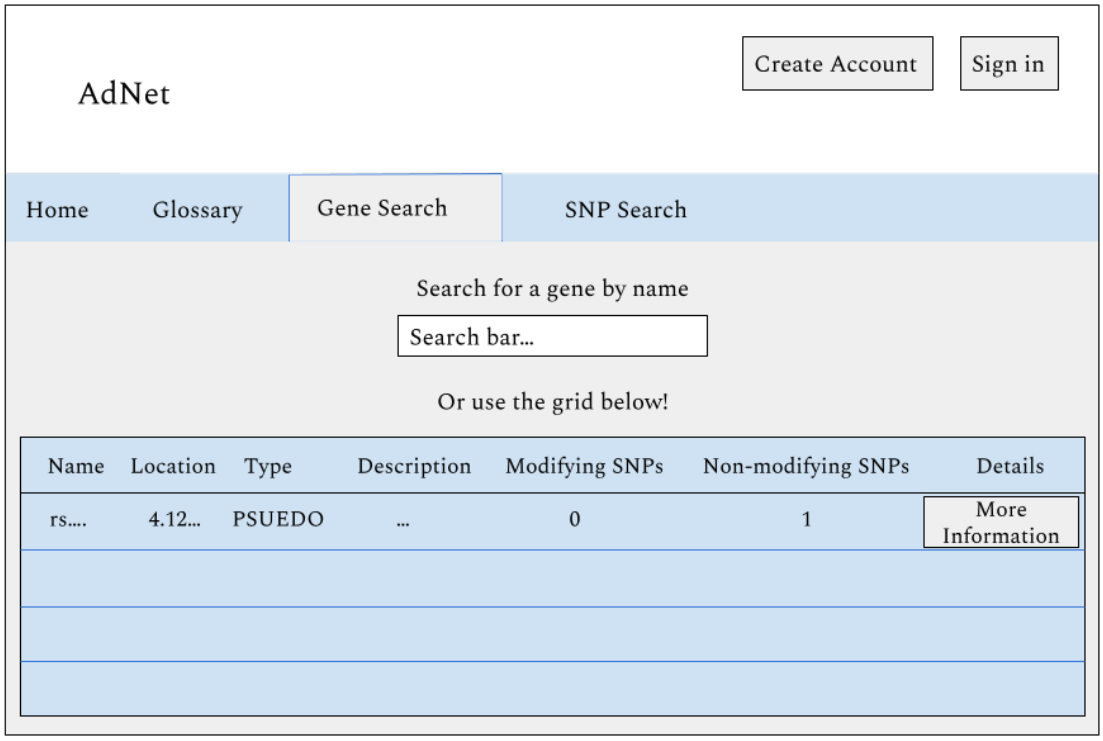

Figure 10. Sample Page Design

there is a basic template which all generated neural nets should follow, an example of which is shown in Figure 11. Because predicting whether an individual will have AD is a simple yes or no outcome, each job is a binary classification problem.
For each SNP that is selected as input for a job, the neural net will take in four input values relating to that SNP. For example, if there are three SNPs in a job, then there will be 12 input features for the neural net. The input features and their data types for each SNP are

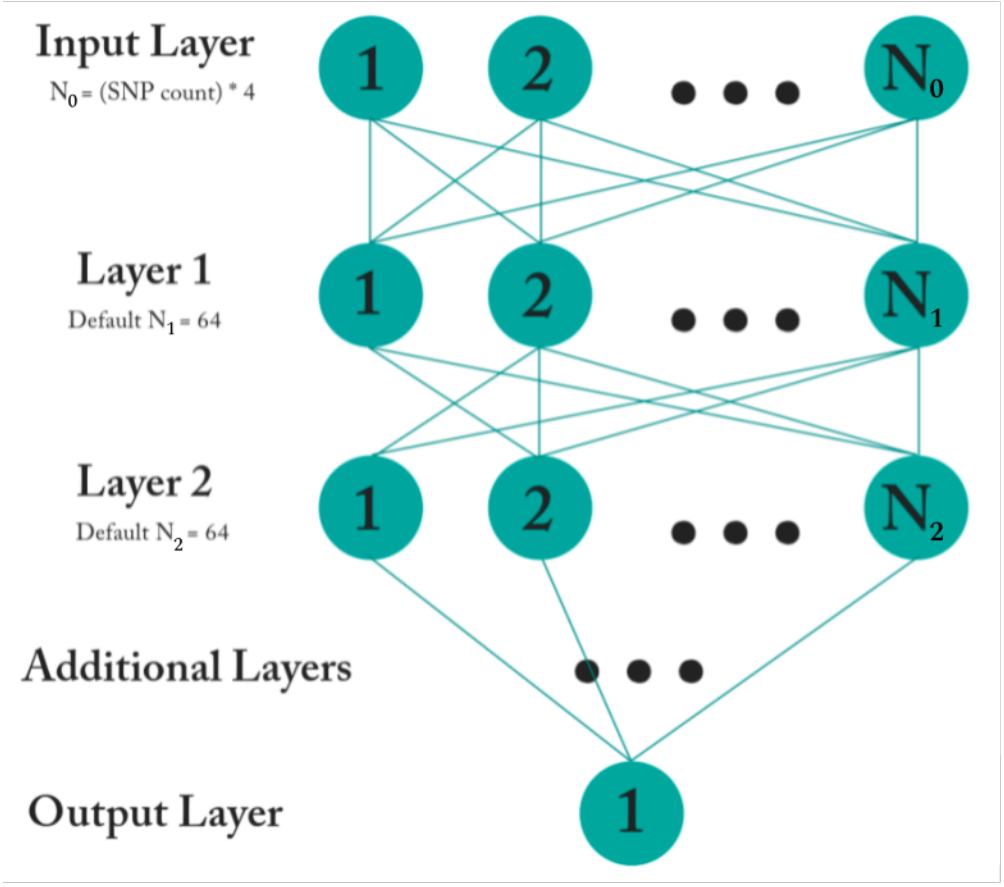

Figure 11. Neural Net Diagram

listed below:

- Functional Class (Categorical)[3]
- Risk (Numeric)
- Location Class (Categorical)
- Chromosome (Categorical)

All SNPs are annotated with a functional class. All variant class definitions and their respective locations can be seen in Figure 12 in the transcript structure. The risk numeric value for each SNP also comes from the respective Ensembl annotation [3]. Ensembl is an open source project which provides a variety of annotations for genomic and proteomic information. One of their key efforts has been to annotate SNP data with the location class, risk level, and functional class of the SNP. As a result, they have richly annotated SNP records, and are a common resource for bioinformatics research. A slightly simplified version of the Ensembl classification diagram is shown in Figure 12.

This simplified diagram depicts a gene laid out from the 5' to 3' end, with various regions annotated accordingly. Each functional class is listed in a particular location. For example, regulatory region and TF binding site classified SNPs take place upstream at the transcription factor binding site of a gene. Thus, both of these functional classes are listed in Location 1. In another example, location 5 represents SNPs that take place in an intron of a particular gene. Each possible functional class for an SNP are defined on the glossary tab of the web application.

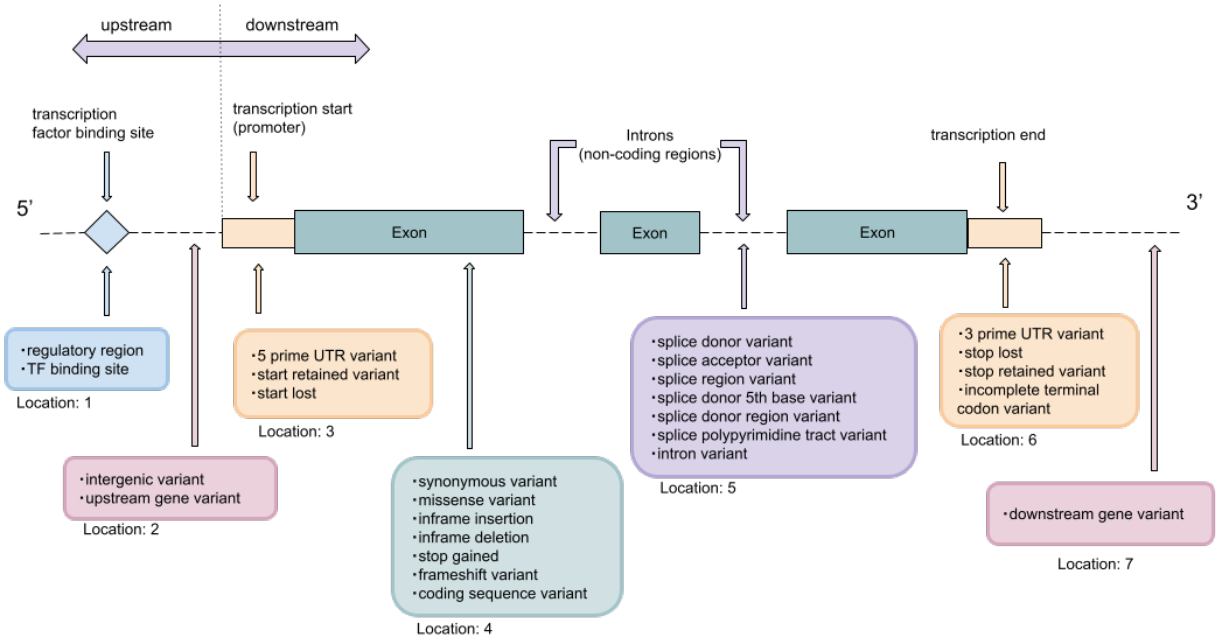

Figure 12. Transcript Structure Annotated with Variant Functional Class and Location[3]

4. Implementation

4.1. Technology

Throughout the project, the primary language used for development was Python, often in combination with bash scripting. For the client-facing web application in particular, a Django framework was used in conjunction with JavaScript, HTML and CSS for the front-end. The primary web application made use of Django's built in SQLite database for user authentication, and a remote MongoDB was used for storing all other data. On the second remote server, TensorFlow was run with a Flask application to generate machine learning models. A remote git repository was used to share code across devices throughout development. Finally, various API's were utilized, such as Ensembl's protein REST API [4], and the GWAS catalog [5] from the National Institute of Health (NIH) for data scraping.

The remote server where the application was deployed was not where implementation and testing took place during sprint cycles. Django conveniently offers its own local development server which works well within a PyCharm IDE (something I already have on my own device). Thus, implementation, testing, and debugging of initial code took place on my local machine via PyCharm. Code was then saved in a remote git repository to simplify continuous integration on the production server. The repository containing project code was cloned on both my device and the eucalyptus server. Once the testing and implementation steps of a sprint cycle were completed, these changes were committed and pushed to the remote git repository. If any changes needed to be made to fix deployment issues, these changes were also committed (from the remote server) and pushed (with the exception of Django configuration files).

4.2. Overview

Implementation of the web tool took place over the course of three general phases. These phases were further broken up into individual sprints to allow for more focused workflows throughout development. Initial sprints focused on pulling and scraping SNP and gene data for the web application to display later on the front-end. During the second phase focus moved to developing the front-end of the proposed web application. The final phase focused on the remote TensorFlow server and visualizing model results. Sprint cycles followed the typical agile development model of five steps: Identifying requirements, planning tasks, implementation, testing, and deployment. Of course, these cycles were adapted depending on the general phase during which they took place. Furthermore, it is worth noting that these rough phases contain lots of overlap, and it was not uncommon throughout development to have a sprint cycle addressing multiple phases at once.

4.2.1. Data Collection Phase

As mentioned before, this phase was primarily focused on gathering genetic data relating to AD. Sprints during this phase deviated slightly from typical sprint steps due to the lack of a deployable web product. Most of these sprints typically focused on obtaining, cleaning, and

storing data from a single API at a time. When requirements were being gathered about data from a particular API, five biology students were consulted to ensure that appropriate information was collected for use. Students were used instead of professors for this stage given their decreased expertise in the field. They were very useful subsequently for helping identify any gaps in understanding that a normal student or researcher might have. These requirements then led to the generation of sprint tasks, examples of which can be seen in Table 3. Next, during the implementation and testing steps, data was collected and checked for validity, but not stored in Mongo. Because nothing was ever going to be "deployed" during these sprints, the final stage of each sprint cycle involved data being uploaded to the remote Mongo database for more permanent storage and easier viewing. Two examples of data collected during this phase can be seen in Tables 1 and 2. In both tables, the MongoDB document fields are presented in the left-hand column, while real example values are displayed on the right.

Field	**Value**
_id	"rs271066"
MAF	0.3562
chr	3
chr_pos	"6336008"
functional_class	"intron_variant"
genes	["MTARC2P1", "LOC12490341", "GRM7-AS3", "LOC10537942"]
is_intergenic	true
location	"3:6336008-6336008"
minor_allele	"T"
pvalue	0.000003
region	"3p26.1"
risk_level	"MODIFIER"
strongest_risk_alleles	[{'risk_freq':"NR", 'allele_name':"rs271066-?", 'allele_value':"?"}]
studies	{'href':"https://www.ebi.ac.uk/gwas/..."}
values	["A", "C", "T"]

Table 1. Example SNP Data

Field	**Value**
_id	“MMP1”
authority	“HGNC”
catalytic_activity	[“Cleavage of the triple helix of collagen at. . . ”]
chromosome	“11”
cofactors	[“Zn(2+)”]
description	“matrix metallopeptidase 1”
expression	{‘adrenal’:“0.0609”, ‘appendix’:“17.6” . . . }
id	4312
identifier	“HGNC:7155”
locations	[“Secreted, extracellular matrix, . . . ”]
mod_len	1
nm_len	0
protein_description	“Interstitial collagenase”
protein_features	[{‘type’: “Signal”, ‘location’: “1-19”, ...}, {‘type’: “Propeptide”, ‘location’: “20-52”, ...}]
range	{‘begin’: 2789919, ‘end’: 2798160, ‘orientation’: ‘minus’}
rna_sequences	[{‘transcript’: “2”, ‘seq’: “ATATTGGAGCAGCAAGAGG”. . . }, {‘transcript’: “1”, ‘seq’: “ATATTGGAGCAGCAAGAGG”. . . }]
sequence	“MHSFPPLLLLLFWGVVSHSFPATLETQEQDVDLVQ. . . ”
snp_effects from HYAL2	[{‘snp’: “rs709210”, ‘amino_acid’: 18, ‘affected_features’: [{‘type’: “Signal”, ‘location’: “1-20”, ‘evidences’: [{‘evidenceCode’: “ECO:0000255”}]}]}]
snp_list	{‘modifying’: [‘rs573521’], ‘non_modifying’: [ ]}
type	“PROTEIN_CODING”

Table 2. Example Gene Data

4.2.2. Web Application Phase

The second phase focused primarily on developing the client-facing Django application, both front-end and back-end, and followed the typical sprint cycle unlike the data collection sprints. Sprint task planning, implementation, testing, and deployment were straightforward during this phase, and examples of sample tasks can be seen in Table 3.

4.2.3. TensorFlow Phase

The third phase was perhaps the most complicated when it came to implementation and validation. Artificial data had to be generated (one of the many sprint tasks during this phase) in order to use for training and testing a TensorFlow model. Implementation was relatively straightforward when it came to sprint tasks, but validation took much more time, and is described in section 5.

4.3. Setbacks

The biggest setback in this project was the inability to access real genomic data for the remote TensorFlow server. This setback did however, come with a silver lining. First, being unable to obtain said data spoke volumes to how seriously researchers regard the need for privacy when it comes to genomic information. Additionally, the lack of real data made it easy to test and verify that the TensorFlow server was creating reasonable models, because I was able to generate my own artificial data with simple predetermined patterns. Presently there are no known combinations of SNPs that allow researchers to predict whether an individual will be symptomatic of AD (hence the point of this project). This meant that if real data had been obtained, more artificial data would likely have been needed anyway to verify that the model could find patterns in the data if they did exist. The lack of real genomic data also reduced the scope of work that was needed to be done on the remote server significantly. Had real genomic data been available, a significant amount of time would have been necessary to extract, clean and format the data into something usable for this application.

Another difficulty arose after the data collection phase of development. When initially generating the user requirements (listed in Section 2), one of the goals was to allow users to select both SNPs and genes for building a neural network. At that time, I had assumed that no more than a few SNPs relating to Alzheimer's disease could be found on a single gene, so this wouldn't increase the number of input features on a model significantly. However, after data collection I conducted some analysis and quickly realized that on some genes, up to 100 SNPs existed that related to AD. With neural net models, it is important to limit the number of input features so as to prevent the model over-fitting to the data. If a lot of specific unique patterns of input features exist, then the trends that the model will identify will not generalize well to the rest of the data. This meant that the requirement of allowing users to select genes for a job was removed, and instead only SNPs could be selected for job configurations. Furthermore, the requirement about suggestions for improving neural network performance was not met. Although lots of information exists in the results section of the web application to help users determine if their job was under-fitting or over-fitting

to the data, examples of specific machine learning configurations or SNP selections were not suggested to users after completion. This was due to the wide scope of potential values (both machine learning and SNPs) available to users for selection, as well as time constraints.

Phase	Sprint Task Example
Data Collection	GWAS API research and preparation for pulling data (Identify AD trait ID(s))
	Pull initial SNP list from GWAS database (EBI)
	Clean initial SNP list and remove SNPs without names, locations, or missing annotations
	Update SNP documents with protein information (if applicable) (Ensembl)
	Configure error handling for missing Ensembl information
Web Development	Create gene grid GUI on front-end
	Create endpoint to return all genes to front-end to populate the grid
	Create gene grid search bar GUI on front-end
	Create endpoint to return all gene names to front-end for search bar
	Create a "More Information" button for each row in grid
TensorFlow	Prepare remote volta server to host TensorFlow flask application
	Create basic initial flask application
	Create endpoint to collect job submission requests from Adnet.app
	Generate artificial data with hidden pattern to validate TensorFlow models
	Create function to generate machine learning model for each job submission

Table 3. Sprint Tasks

5. Testing and Validation

5.1. Data Collection

Given the wide scope of topics and functionality covered in this web tool, validation and testing varied drastically depending on the sprint tasks. During the data collection phase of development, validation included checking that different databases had consistent information about an SNP or gene. Furthermore, validation was also conducted by handling errors gracefully and setting expectations around what type of data needed to be available for an SNP or gene record to be displayed. Required attributes for each type of record are listed below.

1. SNP Record Requirements
    (a) Associated gene(s)
    (b) Location
    (c) Functional Class
    (d) Valid risk level
2. Gene Record Requirements
    (a) Gene ID
    (b) Gene Name
    (c) Chromosome
    (d) Type
    (e) At least one AD-associated SNP, either non-modifying or modifying

Subsequently, the data collection phase focused primarily on validating data rather than testing any front-end functionality for users. Despite this, there were various back-end test cases that were created and utilized to ensure that the data was being set correctly if applicable, as shown in Table 4.

Test Case	Scenario	Input(s)	Expected Output
1	Gene has documented expression values	SPG11	{ “adrenal”: 6.17, “appendix”: 12.3, “bone marrow”: 6.9, “brain”: 8.25, “colon”: 9.36, “duodenum”: 9.01, “endometrium”: 8.01, “esophagus”: 7.84, “fat”: 7.68, “gall bladder”: 8.87, “heart”: 4.99, “kidney”: 9.1, “liver”: 5.2, “lung”: 7.48, “lymph node”: 9.4, “ovary”: 5.93, “pancreas”: 1.64, “placenta”: 7.54, “salivary gland”: 3.49, “skin”: 7.35, “small intestine”: 8.07”, “spleen”: 9.59, “stomach”: 8.49, “testis”: 13.0, “thyroid”: 18.1, “urinary bladder”: 8.63 }
2	Gene does not have documented expression values	LOC100420187	{ }

Table 4. Test cases for sprint task: Update gene data with expression values from NCBI database

5.2. Web Application

Once sprints began to focus on the Django web application functionality, more testing was included with each sprint task. An example of some sample test cases for a sprint task can be seen in Table 5. There were generally more test cases during these sprints because the possibilities of user inputs increased the likelihood of errors significantly. For example, when trying to simply click the submit job button on the front-end, validation has to occur on the job configuration, and different scenarios relating to the job queue must also be considered (and tested).

5.3. TensorFlow Server

For the remote TensorFlow server, validating job models involved using the artificial generated data that was discussed earlier. When developing the artificial data, I came up with my own mildly complicated “patterns” with a few SNPs that I thought would lead to an accurate prediction model. For example, if one SNP existed with a higher risk level, unless a particular specific SNP was present in the record, then the outcome would lead to AD no matter what. I did test multiple patterns, but the general structure of validation with each pattern was similar. Once I had a pattern developed logically, I then generated approximately 3,200 records which followed each scenario in said pattern with 100% accuracy. Once I made sure that the neural net model was achieving complete accuracy during training and validation, I then added another 200 artificial records which did not follow the pattern intentionally. If I saw a slight decrease in accuracy once this new data was added, I felt reasonably confident the model was capturing patterns correctly. Because I knew there was at least some hint of a pattern to be found, this allowed me to implement and check the validity of the TensorFlow models when I entered those specific SNPs. Finally, I tested

Test Case	Scenario	Input(s)	Expected Output
1	Job creation tab clicked, no name typed, update button clicked	Tab click, no name, button click	Row is not created, validation message appears
2	Job creation tab clicked, unique name typed, update button clicked	Tab click, unique name, button click	Empty configuration created and grid updated
3	Job creation tab clicked, non-unique name typed, update button clicked	Tab click, non-unique name, button click	Validation error, grid not updated
4	Job creation tab clicked, non-unique name typed, unique name typed, update button clicked	Tab click, non-unique name, button click, unique name, button click	Validation error, then empty configuration created and grid updated
5	Job creation tab clicked, unique name typed, valid SNP entry typed	Tab click, unique name, valid SNP entry, button click	Job configuration created with SNP, grid updated
6	Job creation tab clicked, unique name typed, invalid SNP entry typed	Tab click, unique name, invalid SNP entry, button click	Validation error, grid not updated
7	Job creation tab clicked, unique name typed, cancel button clicked	Tab click, unique name, cancel button click	New row deleted
8	Job creation tab clicked, non-unique name typed, cancel button clicked	Tab click, non-unique name, cancel button click	New row deleted

Table 5. Test cases for sprint task: Design job configuration creation GUI

random SNPs with the data that was fit to an original pattern to make sure the accuracy of the model decreased (because the rest of the SNP data was randomized).

It is important to note that this validation method, as with most validation methods for machine learning, was limited. While the accuracy and loss values seemed reasonable, and it didn't appear that the initial default model was over-fitting to the data, it is difficult to determine if those results would generalize well to actual genomic data sets. However, because users are allowed to customize their machine learning model specifications, should over-fitting or poor accuracy results be generated from a model, users are able to update and modify their job configurations to mitigate these problems. Examples of test cases from this phase of development can be seen in Table 6.

Test Case	Scenario	Input(s)	Expected Output
1	Submit button is hit on a job with no values	Click job configuration tab, click submit on empty configuration	Alert tells user to add SNPs to the configuration prior to submitting it.
2	Submit button is hit on an invalid job configuration	Click job configuration tab, click submit on invalid configuration	Validation error appearing under the incorrect field that is making the job invalid
3	Submit button is hit on a valid job configuration, and there are no other jobs in the queue	Click job configuration tab, click submit on valid configuration	Alert notifies the user of submitting the job successfully. Status of the job is updated to running
4	Submit button is hit on a valid job configuration, and there are other jobs in the queue	Click job configuration tab, click submit on valid configuration	Alert notifies the user of submitting the job successfully. Status of the job is updated to pending with the queue index

Table 6. Test cases for sprint task: Create an endpoint for submitting a job to the queue

With regards to validation, TensorFlow thankfully has built in functionality to display model architecture [12]. This made visualization and validating model structures relatively straightforward. Figure 13 is an example of a diagram that captures the overall architecture of a neural net model that was produced. The top row of the diagram represents the input layer, followed by hidden layers and the final output layer at the bottom. With each layer, there is a set of parentheses which provides important information regarding the shape of the model. The first number in the parentheses indicates the batch size used in the layer. Batch size refers to how many training examples are processed together in each training iteration. In the input layer specifically, the second number in the parentheses indicates the shape or number of input features that the model expects. So if the input layer looks like (32, 8), that means there is a batch size of 32, and 8 input features which have been submitted to the layer. In the hidden layers and output layer, the second number in the parentheses indicates the number of neurons in each layer. For example, if a hidden layer has an input value of (32, 8) and an output value of (32, 64) this means that there are 64 neurons in this layer (and a batch size of 32). In each job, default values are provided for the neural net model shape, however these values can be changed to your specifications (and those changes will be reflected in this diagram). This makes validation of neural net architecture easy, because the configuration of the model set by the user and the visual diagram are easily compared.

Once model architecture was validated, the next goal was to validate the accuracy of the model that was generated. This was done using a plot chart that takes in the history of the TensorFlow model, and depicts the changes in accuracy and loss values over time graphically.

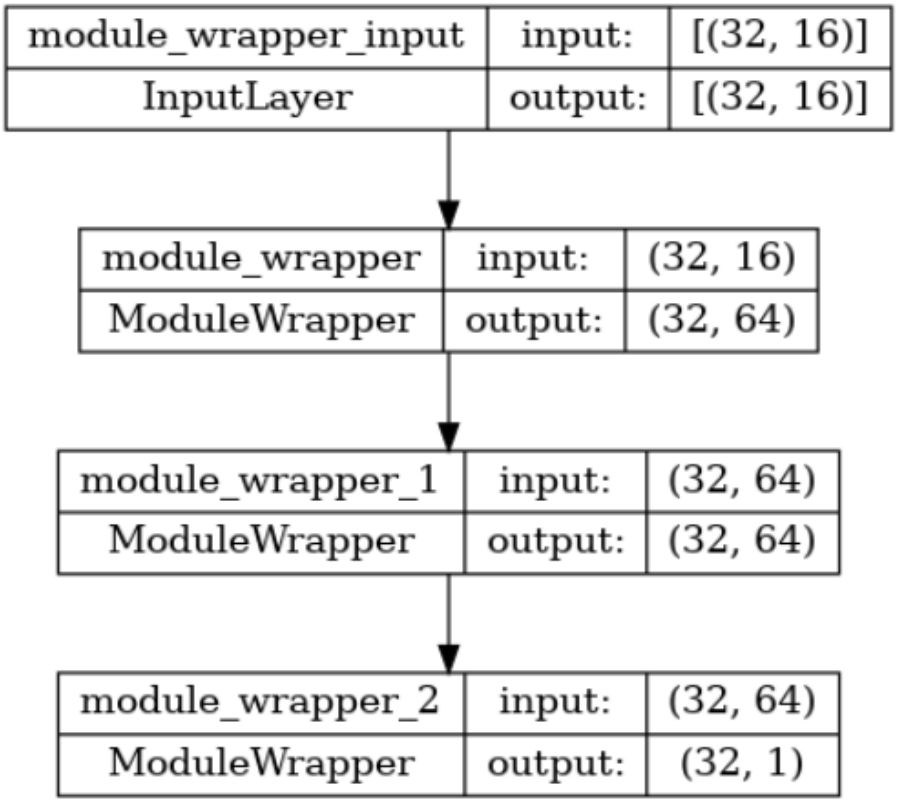

Figure 13. Architecture Validation Diagram

Initially during this validation, data was used that led to 100% accuracy when it came to model prediction during both the training and validation phases, to ensure that the model could find basic patterns. Once this accuracy was established, then more and more "noisy" data was incorporated to reduce accuracy slightly.

An example of the results during a run that incorporated noisy data can be seen in Figure 14. The two graphs in this figure provide further insight on the model performance. The left diagram displays the overall accuracy of the model during training and validation phases. Perfect accuracy, indicating the model always guesses correctly, is 1.0. On the right, the loss of the model is graphed as well. Loss is a fundamental metric which measures the differences between a model's predictions and actual target values. In supervised learning, the goal is to minimize loss. Initially, the loss should be typically high because the model's weights are random, and its predictions are far from accurate. Looking at loss performance is essential because if the loss on the training data continues to decrease while the loss on a separate validation dataset starts increasing, it may indicate overfitting. Overfitting occurs when the

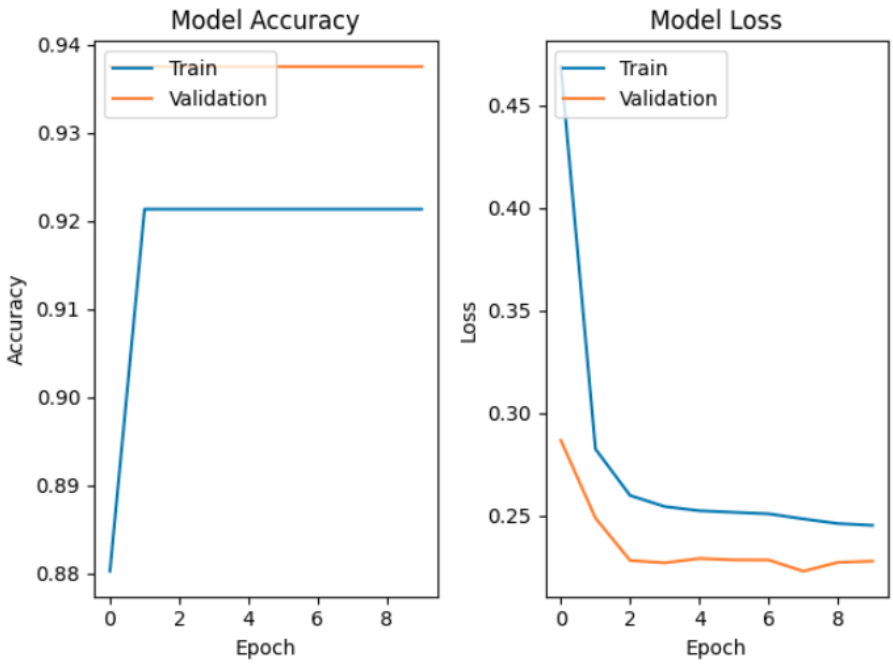

Figure 14. Model Accuracy Diagram

model learns to fit the training data too closely and performs poorly on unseen data. On the other hand, if both the training and validation loss remain high or decrease too slowly, it may suggest underfitting, indicating that the model is too simple to capture the data's complexities. The loss and model performance charts in Figure 14 indicated a relatively accurate run without overfitting or underfitting the data. This test was run multiple times with various data and SNP patterns to ensure consistency with model accuracy and loss.

Another important visualization tool which helped validate the neural net models was a confusion matrix [7]. A confusion matrix, an example of which is shown in Figure 15, provides a detailed breakdown of the model's predictions and accuracy. The top left quadrant indicates the number of true negatives in the model. True negatives are instances where the model predicted a negative output and was correct (i.e. the model predicts no AD, and the individual actually doesn't have AD). The bottom right box indicates the number of true positives, or cases where the model accurately predicts a positive output. In a classification model, high values in the top left and bottom right boxes are desired, because they indicate high accuracy of the model as a whole. Conversely, the top right box indicates the number of false positives: cases where the model predicted an output that was positive, when the real result was negative (i.e. the model predicts someone has AD when they actually do not). The bottom left box represents false negatives: cases where the model predicted that an output was negative when the result was actually positive. Low values in these boxes are desired, because they indicate relatively high accuracy and a low number of errors generated by the model. This visualization is useful to see if the classification model is relatively accu-

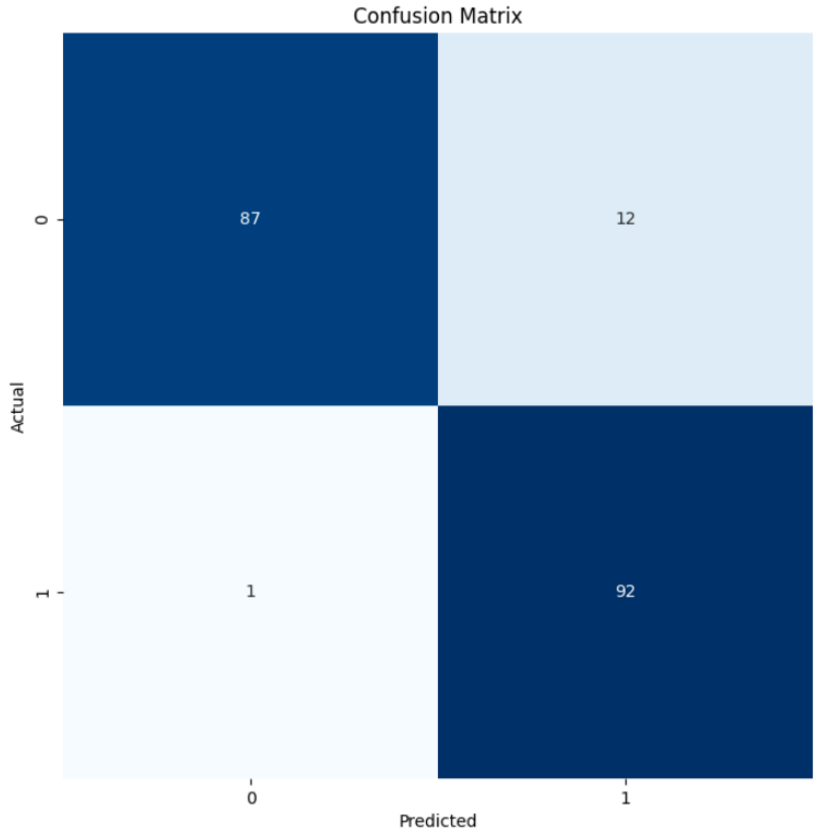

Figure 15. Confusion Matrix Diagram

rate and balanced, or if it skews heavily towards false positives or negatives. For example, if there are lots of false positives, and no false negatives, this could imply that the model is overly optimistic or aggressive in classifying instances as positive. As one can see in Figure 15 this particular model was relatively balanced with strong accuracy, indicating a relatively successful model (which was expected).

A final visualization that was used to validate neural net models was a feature importance diagram. This diagram, an example of which can be seen in Figure 16, displays which features of the data had the most impact in the first layer of the model. It is important to note, after the first layer it becomes difficult to see which features have the most impact later on, so these weights may change over the course of the model. However, it can still provide some insight into which features and attributes of the data may have more impact on the prediction of AD. This was helpful in validating neural net models, as it allowed some insight into which factors had a stronger impact on the accuracy of the model. This was easy to compare to the pattern that was set in the artificial data, so it made validation of the model simple. In the example seen in Figure 16, we can see that the first SNP's risk level played a much more significant role in the model development than other input features. This might suggest to a user that they should incorporate other SNPs with similar risk levels in future

jobs.

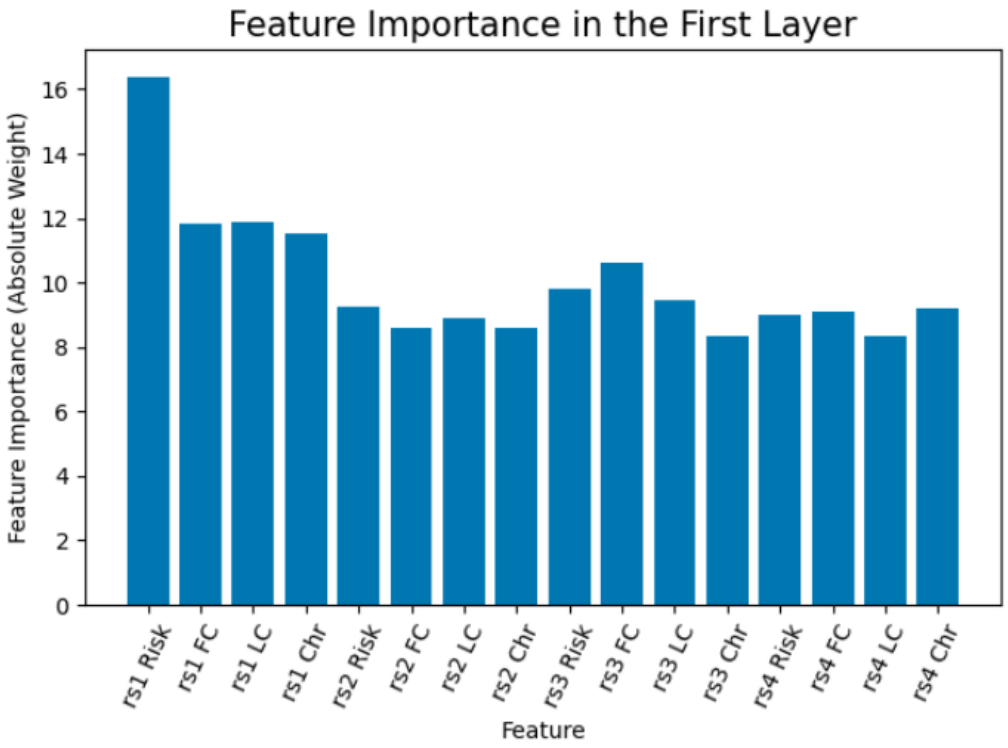

Figure 16. Feature Importance Diagram

Further testing was done to ensure that the TensorFlow application itself could handle multiple job requests with ease. To do so, a testing script which generated a series of fake job requests from users was developed and run multiple times during testing. This process took a significant amount of time, but was worthwhile to ensure that the TensorFlow application could handle a large number of simultaneous requests with ease.

6. Deployment

Deployment of both the web application server and the TensorFlow server involved numerous steps to provide full functionality for clients.

6.1. Web Application Deployment

The main client-facing web application is available through a web browser. This was made possible through Apache on a remote Eucalyptus server, which was configured to host the base Django application. As discussed in the implementation section, initial implementation and testing of new code did not take place on the remote server, but rather on my personal machine. Once code from a sprint was ready for deployment, changes to the project were committed and pushed to my remote git repository. Changes were then pulled (via git) on the remote server, and the Apache service was reloaded so that those changes would be displayed.

One area where I did wish to gain experience was deploying an application to a valid domain name. Having never done so prior, the process of configuring an Apache server to host a domain name was an educational experience. General steps followed for deploying the Django application to a valid domain name can be seen below. The domain name used for hosting the web application, "adnet.app", was purchased from Google Cloud domains.

1. Purchase domain name
2. Generate DNS records
    - A (IpV4) record
    - CNAME record
3. Generate SSL Certificate
4. Configure Apache file to use SSL certificate and domain name
5. Configure Django settings to use domain name
6. Reload Apache service

For the web application to be deployable on the remote server, Apache, Python 3, and pip (a Python package manager tool) were installed first. Given that implementation and testing of the project took place on my personal machine, it was also essential to incorporate a virtual environment into the project. A Python virtual environment or "venv" allows users to create isolated environments for different Python projects. The venv also simplifies adding and removing various Python packages across multiple devices. Using pip, a requirements.txt file was generated and added to the git repository. Although the virtual environment was not cloned between devices (venvs cannot be shared across devices), the requirements file allowed pip to easily read and install necessary packages in one simple command.

Once the venv was set up for the project, and the Apache files were successfully configured, continuous integration throughout the project was simple and straightforward. Prior to each new redeployment, static files were moved for Apache to access using Django's built-in manage script. As discussed earlier, the main database utilized throughout the project was hosted on MongoDB Atlas, which is a cloud database service. Accessing the database from various devices was simple, since Mongo allows access to the database by IP address.

6.2. TensorFlow Deployment

Given that the TensorFlow application isn't client-facing, deployment of preparing the application on the remote Volta server was much easier. A simple Flask application was created for handling requests to the server, and was quickly deployed without the need for Apache or a domain name. Since there were only a small number of pip packages needed to run this application, a virtual environment was not needed (although a requirements.txt file was still generated for documentation purposes). The TensorFlow package (version 2.8.0) was installed simply via pip, a python package manager.

7. Future Work

The web tool, while functional and educational, could certainly be further improved upon. The primary limitation of this project at the moment is the lack of accessibility outside of the UW La Crosse VPN. If a user who is outside of the university VPN tries to reach adnet.app, the web application simply doesn't load and the page times out. Ideally, I would like to move the web application to a remote server outside of the VPN so that individuals outside of the university could reach and access the site. Furthermore, visualization of the web tool could be improved as well. Current visualizations include displaying the overall model architecture, a confusion matrix, loss and metrics plots, and a feature importance diagram. These visualizations provide decent insight into how the machine learning algorithm performed, but further tools, such as insight into layer activation may also be useful to users in the future.

There are many smaller functionalities I would also like to incorporate in the future, such as direct linking from terms to the glossary page for definitions. In addition, I would obviously love to see real data incorporated into this project, although that may not likely happen in the future. Another idea that may be worth incorporating would be to let users select certain input features of an SNP specifically, so they can even further customize their neural net model. When working on the TensorFlow application, validation and testing of the application required more time than initially expected. There are many different avenues that could be followed to increase user functionality and ease of use, but time constraints prevented new functionality being added during the sprint cycles.

References

[1] "2022 Alzheimer's disease facts and figures." In: *The Journal of the Alzheimer's Association* 18.4 (2022), pp. 700–789. DOI: 10.1002/alz.12638.

[2] M Carmo Carreiras et al. "The multifactorial nature of Alzheimer's disease for developing potential therapeutics." In: *Current topics in medicinal chemistry* 13.15 (2013), pp. 1745–1770. DOI: 10.2174/15680266113139990135.

[3] Ensembl. *Ensembl Variation - Calculated variant consequences*. URL: https://useast.ensembl.org/info/genome/variation/prediction/predicted_data.html.

[4] *Ensembl REST API endpoints*. URL: https://rest.ensembl.org/.

[5] *GWAS Catalog*. URL: https://www.ebi.ac.uk/gwas/.

[6] Project Management Institute. *Scrum or Waterfall: What's the Difference?* URL: https://pmi-macedonia.mk/event/scrum-or-waterfall-whats-the-difference/.

[7] Sarang Narkhede. *Understanding Confusion Matrix*. URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[8] *National Human Genome Institute*. URL: https://www.genome.gov/genetics-glossary/Gene.

[9] Jill Rasmussen and Haya Langerman. "Alzheimer's Disease - Why We Need Early Diagnosis". In: *Degener Neurol Neuromuscul Dis* 9.1 (2019), pp. 123–130. DOI: 10.2147/DNND.S228939.

[10] Francis Robert and Jerry Pelletier. "Exploring the Impact of Single-Nucleotide Polymorphisms on Translation." In: *Frontiers in Genetics* 9 (2018), p. 507. DOI: 10.3389/fgene.2018.00507.

[11] Barker S. Shastry. "SNPs in disease gene mapping, medicinal drug development and evolution." In: *Journal of Human Genetics* 52 (2007), pp. 871–880. DOI: 10.1007/s10038-007-0200-z.

[12] TensorFlow. *TensorFlow v2.13.0 Plot Model*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/utils/plot_model.